EL835823874US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

for

METHODS AND APPARATUS FOR REAL-TIME BUSINESS VISIBILITY USING PERSISTENT SCHEMA-LESS DATA STORAGE

10051516.102601

Inventors:

Colin P. Britton, a citizen of the United Kingdom,
residing at 17 Pheasant Lane,
Lexington, Massachusetts 02421

Amir Azmi, a citizen of India,
residing at 158 Concord Road, Apt. # K-09,
Billerica, Massachusetts 01821

Ashok Kumar, a citizen of India,
residing at 447 Boston Post Road East, Apt. #96,
Marlborough, Massachusetts 01752

Noah W. Kaufman, a citizen of the United States,
residing at 66C Hampshire Street,
Cambridge, Massachusetts 02139

Chandra Bajpai, a citizen of the United States,
residing at 21 Pamela Road,
Natick, Massachusetts 01760

Robert F. Angelo, a citizen of the United States,
residing at 4 Southwest Circle,
Sudbury, Massachusetts 01776

David Bigwood, a citizen of the United Kingdom,
residing at 324 Concord Avenue,
Lexington, Massachusetts 02421

EL835823874US

Background of the Invention

This application claims the benefit of United States Provisional Patent Application Serial No. 60/324,037, filed September 21, 2001, entitled “Methods and Apparatus for Real-Time Business Visibility Using Persistent Schema-Less Data Storage” and U.S. Provisional Patent Application Serial No. 60/291,185, filed on May 15, 2001, entitled “Methods and Apparatus for Enterprise Application Integration.” This application is also a continuation-in-part of United States Patent Application Serial No. 90/917,264, filed July 27, 2001, entitled “Methods and Apparatus for Enterprise Application Integration.” The teachings of all of the foregoing applications are incorporated herein by reference.

The invention pertains to digital data processing and, more particularly, to methods and apparatus for enterprise business visibility and insight using real-time reporting tools.

It is not uncommon for a single company to have several database systems – separate systems not interfaced – to track internal and external planning and transaction data. Such systems might of been developed at different times throughout the history of the company and are therefore of differing generations of computer technology. For example, a marketing database system tracking customers may be ten years old, while an enterprise resource planning (ERP) system tracking inventory might be two or three years old. Integration between these systems is difficult at best, consuming specialized programming skill and constant maintenance expenses.

A major impediment to enterprise business visibility is the consolidation of these disparate legacy databases with one another and with newer e-commerce databases. For instance, inventory on-hand data gleaned from a legacy ERP system may be difficult to combine with customer order data gleaned from web servers that support e-commerce (and other web-based) transactions. This is not to mention difficulties, for example, in consolidating resource scheduling data from the ERP system with the forecasting data from the marketing database system.

An object of this invention is to provide improved methods and apparatus for digital data processing and, more particularly, for enterprise business visibility and insight (hereinafter, "enterprise business visibility").

A further object is to provide such methods and apparatus as can rapidly and accurately retrieve information responsive to user inquiries.

A further object of the invention is to provide such methods and apparatus as can be readily and inexpensively integrated with legacy, current and future database management systems.

A still further object of the invention is to provide such methods and apparatus as can be implemented incrementally or otherwise without interruption of enterprise operation.

Yet a still further object of the invention is to provide such methods and apparatus as to facilitate ready access to up-to-date enterprise data, regardless of its underlying source.

Yet still a further object of the invention is to provide such methods and apparatus as permit flexible presentation of enterprise data in an easily understood manner.

Summary of the Invention

The aforementioned are among the objects attained by the invention, one aspect of which provides a method for enterprise business visibility that transforms transactional and other information from a plurality of legacy and other databases in resource description framework (RDF) syntax. This information can be time-stamped (e.g., with expiration dates) and stored in a central database, referred to below as a "holographic" data store.

Further aspects of the invention provide such methods for real-time enterprise business visibility and insight that uses software ("connectors") that can be instantiated via downloading (e.g., using Java® or other such technologies) to provide interfaces to respective disparate database systems. The databases systems may comprise any variety of now or heretofore known systems, e.g. SAP, Oracle, and so forth.

The connectors can, for example, translate between a native language (or Application Program Interface ("API")) of the respective database systems and an internal language/protocol of the enterprise business visibility system. To this end, the connectors can utilize a scripting language to access the respective database systems.

The connectors, according to further aspects of the invention, can query the respective database systems, using simple queries and/or using data mining techniques, based on requests received from the holographic data store and/or from a framework server, a user or otherwise. In related aspects, the data store is periodically updated via application of queries to the database systems.

Further aspects of the invention provide methods as described above in which a graph generator generates directed graphs from the RDF triples in the holographic store. The graphs can be "walked" in order to discern answers to queries for information reflected by triples originating from data in one or more of the databases, and the confidence levels of those answers based in part, for example, on the time-stamps of the underlying triples. In related aspects of the

invention, answers to queries are discerned by applying genetic algorithm-based search techniques to the holographic store.

Another aspect of the invention provides methods as described above in which a framework server accepts queries, e.g., from a user, and formats them for application to the holographic data store.

Further aspects of the invention provide real-time enterprise business visibility and insight that operate in accord with the foregoing.

These and other aspects of the invention are evident in the drawings and in the description that follows.

Brief Description of the Drawings

The foregoing features of this invention, as well as the invention itself, may be more fully understood from the following detailed description of the drawings in which:

Figure 1 depicts an improved enterprise business visibility and insight system according invention;

Figure 2 depicts operation of a software interface "connector" according to the invention;

Figure 3 depicts data flow within a connector according to the invention; and

Figure 4 depicts a directed graph representing data triples of the type maintained in a data store according to the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a real-time enterprise business visibility and insight system according to the invention. The illustrated system 100 includes connectors 108 that provide software interfaces to legacy, e-commerce and other databases 140 (hereinafter, collectively, "legacy databases"). A "holographic" database 114 (hereinafter, "data store" or "holographic data store"), which is coupled to the legacy databases 140 via the connectors 108, stores data from those databases 140. A framework server 116 accesses the data store 114, presenting selected data to (and permitting queries from) a user browser 118. The server 116 can also permit updates to data in the data store 114 and, thereby, in the legacy databases 140.

Legacy databases 140 represent existing (and future) databases and other sources of information in a company, organization or other entity (hereinafter "enterprise"). In the illustration, these include a retail e-commerce database (e.g., as indicated by the cloud and server icons adjacent database 140c) maintained with a Sybase® database management system, an inventory database maintained with an Oracle® database management system and an ERP database maintained with a SAP® Enterprise Resource Planning system. Of course, these are merely examples of the variety of databases or other sources of information with which methods and apparatus as described herein can be used. Common features of illustrated databases 140 are that they maintain information of interest to an enterprise and that they can be accessed via respective software application program interfaces (API) or other mechanisms known in the art.

Connectors 108 serve as an interface to legacy database systems 140. Each connector applies requests to, and receives information from, a respective legacy database, using that database's API or other interface mechanism. Thus, for example, connector 108a applies requests to legacy database 140a using the corresponding SAP API; connector 108b, to legacy database 140b using Oracle API; and connector 108c, to legacy database 140c using the corresponding Sybase API.

In the illustrated embodiment, these requests are for purposes of accessing data stored in the respective databases 140. The requests can be simple queries, such as SQL queries and the like (e.g., depending on the type of the underlying database and its API) or more complex sets of queries, such as those commonly used in data mining. For example, one or more of the connectors can use decision trees, statistical techniques or other query and analysis mechanisms known in the art of data mining to extract information from the databases.

Specific queries and analysis methodologies can be specified by the holographic data store 114 or the framework server 116 for application by the connectors. Alternatively, the connectors themselves can construct specific queries and methodologies from more general queries received from the data store 114 or server 116. For example, request-specific items can be "plugged" into query templates thereby effecting greater speed and efficiency.

Regardless of their origin, the requests can be stored in the connectors 108 for application and/or reapplication to the respective legacy databases 108 to provide one-time or periodic data store updates. For example, connectors can use expiration date information to determine which of a plurality of similar data to return, or if a date is absent, the connector can return the data with a lower confidence level.

Data and other information (collectively, "messages") generated by the databases 140 in response to the requests are routed by connectors to the holographic data store 114. That other information can include, for example, expiry or other adjectival data for use by the data store in caching, purging, updating and selecting data. Those messages can be cached by the connectors 108, though, they are preferably immediately routed to the store 114.

The software connectors 108 may reside on any digital data processing system(s) that is (are) in communications coupling – e.g., via a dial-up connection, bus, cable, network and/or Internet (as indicated by cloud icons), or otherwise -- with the respective legacy databases 140 and with the holographic data store 114. Typically, the connectors reside on computers within

the firewall (or other security barrier) of the enterprise, though, they may reside elsewhere (e.g., local to the holographic store 114 and/or the framework server 116).

In a preferred embodiment, the connectors are implemented as automatically compiling components in a J2EE framework, or the like. Depending on the legacy database from which information is being collected, on the type of query, and so forth, general- or specific-purpose connector modules can be electronically downloaded or otherwise remotely updated as required. Of course, the connectors can be implemented in software other than as components of a J2EE framework and can be configured to communicate with legacy peripheral and other hardware devices.

In embodiments, such as that illustrated here, wherein the connectors 108 are implemented as automatically compiling components in a J2EE framework, or the like, those connectors preferably execute within a suitable environment, e.g., utilizing Java virtual machines running scripted Extensible Markup Language ("XML") operating according Extensible Stylesheet Language Transformation ("XSLT") scripts. The use of XSLT scripts allow the connectors to communicate with a variety of database systems by merely downloading the XSLT using any computer readable medium, e.g. disk, electronic download, or CD-ROM.

Referring to Figure 2, the connectors translate between the API (or other interface mechanisms) of the legacy databases 140 and a language/protocol common to the connectors 108, the holographic data store 114 and the framework server 116. In the illustrated embodiment, that common language/protocol is referred to Intelligent Connector Query Language (ICQL). Thus, for example, requests generated by holographic data store 114 and routed to connector 108a in ICQL (or other language/protocol) are converted (or translated or transformed) by that connector into an appropriate API call to legacy database 140a. Likewise, messages generated by that database 140a in response to such a request are converted by the connector 108a back into ICQL (or other language/protocol).

It will be appreciated that other embodiments may use other common languages/protocols for communications between the connectors 108, the holographic data store 114 and/or the framework server 116, such as, by way of non-limiting example, extensible query language ("XQL") presented as a JDBC-like (Java Database Connectivity) API, returning JBDC result sets to a calling web page or Java program. Still further embodiments may not use a common language/protocol at all.

A more complete understanding of the operation of the connectors 108 may be attained by reference to Figure 3, which shows data flow within a connector 300 according to one embodiment of the invention.

Illustrated is a connector 300 utilizing Hypertext Transfer Protocol ("HTTP") as a vehicle to transfer messages (e.g., requests and responses thereto) with holographic data store 114, such as the one illustrated in Figure 1. Each message 302 (e.g., request) originating from the data store 115 is processed by request, match and action modules 304 - 308, as shown. Although these illustrated modules are specific to an extensible server page ("XSP") environment, it will be appreciated by one skilled in the art that other appropriate modules accomplishing essentially the same functions can be implemented in other types of environments.

The message is sent to the connected legacy database, e.g., 140a, using the appropriate API or other interface mechanism. It can be appreciated that the actual transformation sequence is dependent on the type of legacy database system being accessed and the method of communication between the holographic data store and the connector framework.

Messages received by the connector 300 from the legacy database are likewise processed for return to the holographic data store 114. In the illustrated example, a message 318 is received and routed to a generator module 314 which performs a transformation according to an XSP script, and then routes the message to a transformer module 312. The transformer module 302 transforms the data field contained within the message into RDF triple form suitable for the holographic data store 114 to catalog, and assigns a unique Universal Identification Number

("UID") for later conversion into a Universal Resource Locator ("URL") by the data store 114. Finally, the message is routed to a serializer module 310 and transformed for HTTP transfer to the holographic data store 320.

As can be appreciated, each such module transforms the data and passes it along the stream. The exact composition of engines/modules is specific to the underlying database. For example, if the underlying database executes using SQL, then the connector can have different engines than where the underlying database executes SAP. As such, the connectors are developed to exploit specific database query languages and thereby increase the speed and efficiency in translating protocols and seeking data.

Referring back to Figure 1, the holographic data store 114 stores data from the legacy databases 140 and from the framework server 116 as RDF triples. The data store 114 can be embodied on any digital data processing system or systems that are in communications coupling (e.g., as defined above) with the connectors 108 and the framework server 116 capable of supporting Java ® running XML/XSLT as defined above. Typically, the data store 114 is embodied in a workstation or other high-end computing device with high capacity storage devices or arrays, though, this may not be required for any given implementation.

Though the holographic data store 114 may be contained on an optical storage device, this is not the sense in which the term "holographic" is used. Rather, it refers to its storage of data from multiple sources (e.g., the legacy databases 140) in a form which permits that data to be queried and coalesced from a variety of perspectives, depending on the needs of the user and the capabilities of the framework server 116.

To this end, a preferred data store 114 stores the data from the legacy databases 140 in object-predicate-subject form, e.g., RDF triples, though those of ordinary skill in the art will appreciate that other forms may be used as well, or instead. For example, to represent the statement, "Metatomix is located in Waltham," a RDF representation would contain an object containing "Waltham," a predicate containing "location," and a subject containing "Metatomix".

The predicate would be a unique pointer between the subject "Metatomix," and the object, "Waltham," and would have a tag indicating that the predicate is of type "location." In its simplest form, the example statement, is expressed in RDF as follows:

```
<rdf:description about=http://www.metatomix.com>
<location>Waltham</location>
</ref:description>.
```

The data store 114 can maintain the triples in literal form though, preferably, they are maintained as Uniform Resource Identifiers (hereinafter, "URI"). In some embodiments, relational databases are used to store this information. In the illustrated embodiment, however, the data items are stored in three tables using an algorithm that converts the data into unique 64-bit strings, e.g., in the manner of a "hashed with origin" approach. Those tables are maintained in a backing commercial database, such as the type commercially available from Oracle™ or others in the marketplace.

Referring to Figure 4, the illustrated data store 114 can store – by way of non-limiting example -- RDF triples representing data from marketing and/or e-commerce "legacy" databases, thus providing real-time visibility and insight into enterprise data. The figure particularly illustrates triples representing hotel reservation transactions. Each triple comprises a predicate 402, subject 406 and object 408 such that the object 408 is "linked" to its subject(s) 406 via predicate(s) 402 as explained above.

In the illustrated embodiment, each predicate 402 is assigned a URI 410 such that related data is located via URIs in a hierarchical ordering, represented for example by the directed arrow 402. If the triple is high-level 408 its URI 404 points to a lower set of triples 412, each of which has a URI 414 that may point to data or to further triples 416.

Each subject 406 contains transactional information pertaining to an enterprise resource item, e.g. credit card type, type of product bought or date. For example, as illustrated in Figure 4, a typical subject 420 shows a value of "date of departure" related to a hotel booking

transaction. It can be appreciated from one in the art that many different types of data may be contained within the subject, e.g. literal values, referenced values or additional URI's.

An object 408 contains information pertaining to the "who" of the transaction, such as the person or enterprise initiating the transaction. The object, similar to the subject, may be a literal, e.g. "Smith", or a unique identifier such as a locator address 422 such that each related predicate and subject can be referenced through the object. The identifiers are assigned by the store 114, in sequence, as each is received from the connectors. Portions of each identifier can be coded to reflect which connector the triple was received from and/or the store from which the underlying data was generated.

According to one embodiment of the invention, additional information can be stored, e.g., along with each RDF triple or on a document object basis. This can include expiry date or other adjectival data passed to the hologram store 114 by the connectors 300, e.g., in document request headers or the like, along with data collected from the legacy databases. It can also include tags added by a caching component (not shown) of the data store to reflect version numbers and Universal Unique Identifiers (UUID) or other serial numbers. This expiry and other additional information can be utilized by the hologram data store to use for caching, purging, updating and selecting collected documents and the contained triples, e.g., an internal scheduler and a determination of confidence levels as described below.

The expiry can be of various types, e.g., absolute date/time expiration or relative time expiration. One example among many is that the data expires if it has not been accessed within three hours. A data structure can also be marked as "never expires," applicable for example for a social security number. The data is also tagged with a confidence level, determined through statistical methods, to indicate its validity compared with other like data.

An internal scheduler maintains stored RDF data and deletes data that has expired according to a set of rules. Any RDF triple with a relative expiration time stamp will be deleted, for example, if the data has not been accessed for the time period specified when the data was

created. System Administrators may choose not to delete old data when it expires, essentially forming a large virtual warehouse that grows larger as more data is entered. In this embodiment, the scheduler will not delete data, but rather marks the data as "stale." The confidence tag is reduced to indicate a lower confidence level forthwith, and is returned along with the data to the program requesting the data.

When a search request is applied against to the data store 114, functionality in the data store 114 checks internal data structures (stored in the data store) first. If the data is not found, a command is issued to the appropriate connector to obtain the requested data from the legacy database systems. If the latest data is not available, the most recently stored data (if available) is returned with a reduced confidence factor to indicate any expiration. In a preferred embodiment, illustrated data store 114 polls the legacy database systems 140 (via connectors 108) to obtain current information at pre-determined intervals, times or otherwise.

It can be appreciated that any given transaction (or other event that gives rise to triples of the type stored in the data store 114) may be reflected in multiple legacy database systems 140. When those systems are queried by the connectors, this may result in multiple triples causing redundant or related information to be stored within the holographic store 114. A feature of the data store is the ability to form associations between different sources of data entering the data store through use of a "relationizer" module.

This can be performed by comparing sequential levels of objects and merging triples and documents (or collections) of similar objects. For example, two people at the same address and same last name may be merged into a "family" document (or collection), and so on. (Data in a triple suggesting that these might not be members of the same family could be ignored, for example, if that triple is aged or nearly expired.) The relationizer can be tailored to satisfy end-user's requirements using probabilistic, fuzzy matching and transformation rules. In this way, data storage is both minimized and related such that queries can be executed using the minimal execution time. The data store 114 can also remove redundant information from the legacy databases 140 in a similar manner dependent on the capabilities of the specific database.

A "data crusher" module further reduces the granularity or "size" of the data as time progresses and when individual datum becomes less important. The data crusher operates in tandem with the relationizer as part of a regular data store maintenance routine.

A data mining module utilizes conventional data mining techniques to gather information from the data store 114. The module operates in batch mode or incremental mode, depending on the algorithm selected. *See,* for example, "Iterative Dichotomizer Tree Induction Algorithm," J. Ross Quinlan, Induction of Decision Trees, Journal of Machine Learning, Vol. 1, pgs 81-106, 1986.

The data store 114 is provided with a Web Distributed Authoring and Versioning ("WebDAV") interface to allow for documents to be added, updated and deleted from the database using secure WebDAV client tools, e.g., Microsoft Office, XML Spy or other such tools available from a variety of vendors. Additionally, or in the alternative, a system user can drag and drop onto Web folders in the Microsoft Windows Explorer environment. The interface stores presented document files in a relational database, while parsing out and storing RDF triples in the triples storage area. The interface presents the user with a view of a traversable file system. Files can be opened directly in XML editing tools or from Java programs supporting WebDAV protocols, or from processes on remote machines via any HTTP protocol on which WebDAV is based. Additionally, a WebDAV security system is utilizes to preserve data integrity and protect enterprise business data from unauthorized access.

Moreover, as noted above, the data store 114 further supports a SQL-like query language called hXQL. This allows retrieval of RDF triples matching defined criteria. The hXQL interface is presented as a JDBC-like API, returning JDBC ResultSets to a calling web page or Java program.

The data store 114 includes a graph generator (not shown) that uses RDF triples to generate directed graphs in response to queries (e.g., in ICQL form) from the framework server

116. These may be queries for information reflected by triples originating from data in one or more of the legacy databases 140 (one example might be a request for the residence cities of hotel guests who booked reservations on account over Independence Day weekend, as reflected by data from an e-Commerce database and an Accounts Receivable database). Such generation of directed graphs from triples can be accomplished in any conventional manner known the art (e.g., as appropriate to RDF triples or other manner in which the information is stored). Directed graphs generated by the data store are passed back to the server 116 for presentation to the user.

According to one practice of the invention, the data store 114 utilizes genetic, self-adapting, algorithms to traverse the RDF triples in response to queries from the framework server 116. Though not previously known in the art for this purpose, such techniques can be beneficially applied to the RDF database which, due to its inherently flexible (i.e., schema-less) structure, is not readily searched using traditional search techniques. To this end, the data store utilizes a genetic algorithm that performs several searches, each utilizing a different methodology but all based on the underlying query from the framework server, against the RDF triples. It compares the results of the searches quantitatively to discern which produce(s) the best results and reapplies that search with additional terms or further granularity.

Referring back to Figure 1, the framework server 116 generates requests to the data store 114 (and/or indirectly to the legacy databases via connectors 108, as discussed above) and presents information therefrom to the user via browser 118. The requests can be based on ICQL requests entered directly by the user though, preferably, they are generated by the server 116 based on user selections/responses to questions, dialog boxes or other user-input controls. In a preferred embodiment, the framework server includes one or more user interface modules, plug-ins, or the like, each for generating queries of a particular nature. One such module, for example, generates queries pertaining to marketing information, another such module generates queries pertaining to financial information, and so forth.

In some embodiments, queries to the data store are structured on a SQL based RDF query language, in the general manner of SquishQL, as known in the art.

In addition to generating queries, the framework server (and/or the aforementioned modules) "walks" directed graphs generated by the data store 114 to present to the user (via browser 118) any specific items of requested information. Such walking of the directed graphs can be accomplished via any conventional technique known in the art. Presentation of questions, dialog boxes or other user-input controls to the user and, likewise, presentation of responses thereto based on the directed graph can be accomplished via conventional server/browser or other user interface technology.

In some embodiments, the framework server 116 permits a user to update data stored in the data store 114 and, thereby, that stored in the legacy databases 140. To this end, changes made to data displayed by the browser 118 are transmitted by server 116 to data store 114. There, any triples implicated by the change are updated and forwarded to the respective legacy databases 140, which utilize the corresponding API (or other interface mechanisms) to update their respective stores.

In some embodiments, the server 116 can present to the user not only data from the data store 114, but also data gleaned by the server directly from other sources. Thus, for example, the server 116 can directly query an enterprise website for statistics regarding web page usage, or otherwise.

A further understanding of the operation of the framework server 116 may be attained by reference to the appendix filed with United States Patent Application Serial No. 90/917,264, filed July 27, 2001, and entitled "Methods and Apparatus for Enterprise Application Integration," which appendix is incorporated herein by reference. Likewise, the content of Figure 4 and the subparts hereof may be discerned from a more legible copy thereof filed with aforesaid United States Patent Application Serial No. 90/917,264, filed July 27, 2001, and entitled "Methods and Apparatus for Enterprise Application Integration," which Figure and subparts are incorporated herein by reference

Described herein are methods and apparatus meeting the above-mentioned objects. It will be appreciated that the illustrated embodiment is merely an example of the invention and that other embodiments, incorporating changes to those described herein, fall within the scope of the invention, of which we claim: